50.043 Project Documentation

Collaborators
Lu Jiankun 1002959
Zhao Lutong 1002872
Peng Shanshan 1002974
Gao Yunyi 1002871
Nashita Abd Tipusultan Guntaguli 1003045
Ainul Mardhiyyah 1003115
Hong Pengfei 1002949

Index

How to Run Code

Instructions

You could also refer to Readme of our github
https://github.com/Jiankun0830/ISTD50043_bookReview for setup instruction.

1. Set up the production backend: **one step only!**
    a. Prerequisites:
    pip install boto3 paramiko
    b. Execution: [1 step only]
    Under the root directory of this app:
    python3 production_backend_setup.py
    When executing this script, it will take aws credentials as inputs:

```
Please enter your AWS access key:AKIAWIPE
Please enter your AWS secret access key:(
```

    Reminder: In later part of the execution script, i.e. setting up mongoDB, mySQL may take 3~5 minutes to setup due to the installation, therefore it may looks that it 'hangs' at that stage :)
    When the script finished executing, please wait for 4-5 minus for the server to finish setting up.

    c. Evaluation - To access the web created:
    After running the automation script, you can just view the website by using the IP address displayed on the screen:
    You can find the IP address of our web from any of these places:
        1. The "LC_WEBSERVER_IP"

```
IP dictionary: {'LC_MONGO_IP': '44.230.130.57', 'LC_MYSQL_IP': '44.229.227.10',
'LC_WEBSERVER_IP': '44.230.209.167'}
```

        2. The elastic ip of server

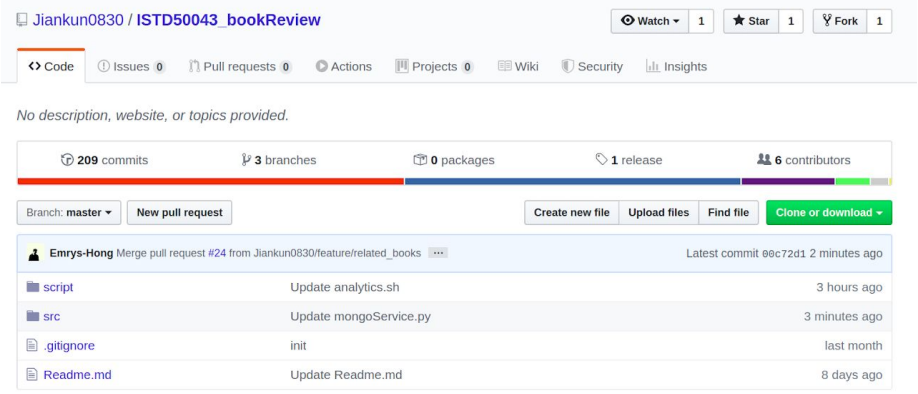
```
Set up server on elastic ip: 44.230.209.167
Step1 git clone web server's code
[]
Step2 run web server's setup script
application_setup.sh
```

        3. The remainder at the end

```
You can view the app though 44.230.209.167 now
```

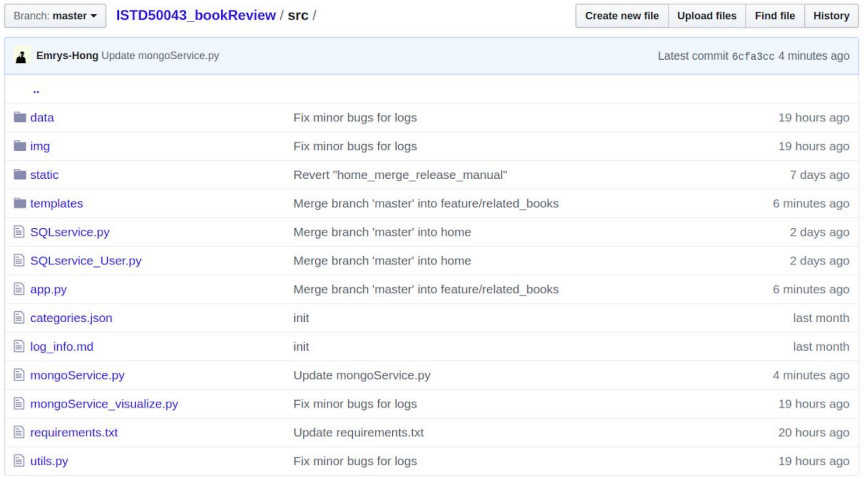
Once we find the IP address for the web, just paste it on the browser, you will automatically be directed to the homepage. e.g. http://44.230.209.167

Git Folder Layout

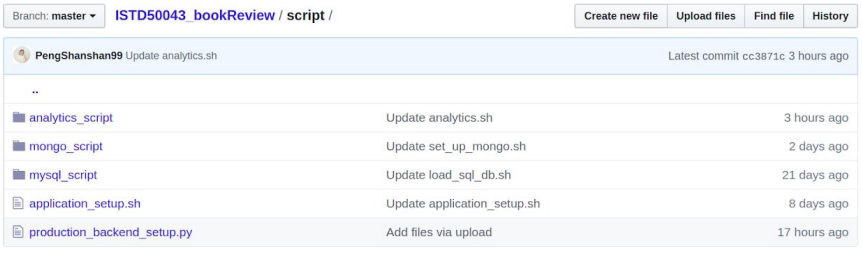


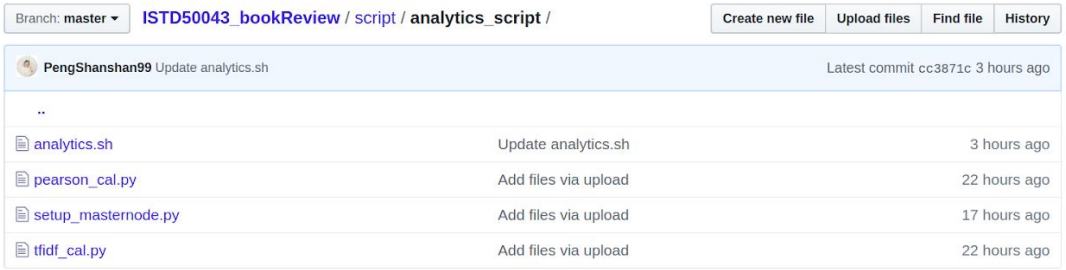
Directory src (stores GOODSHELF app scripts)

App.py (import the Flask module and creating a Flask web server from the Flask module; all the endpoints are defined here)
Directory templates (contains all html pages)
Directory static (contains css javascript functions)
Directory img (contains all images used in the current app)
Directory data (contains all intermediate data files used for analytics of log record)



Directory script (stores automation scripts to set up the app and analytics)

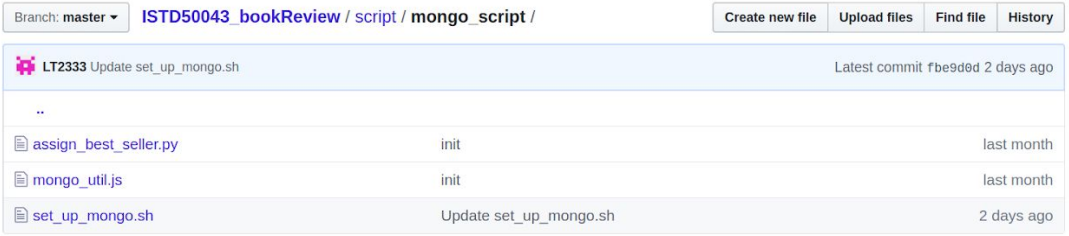




Branch: master ▾ ISTD50043_bookReview / script / analytics_script / Create new file | Upload files | Find file | History

PengShanshan99 Update analytics.sh — Latest commit cc3871c 3 hours ago

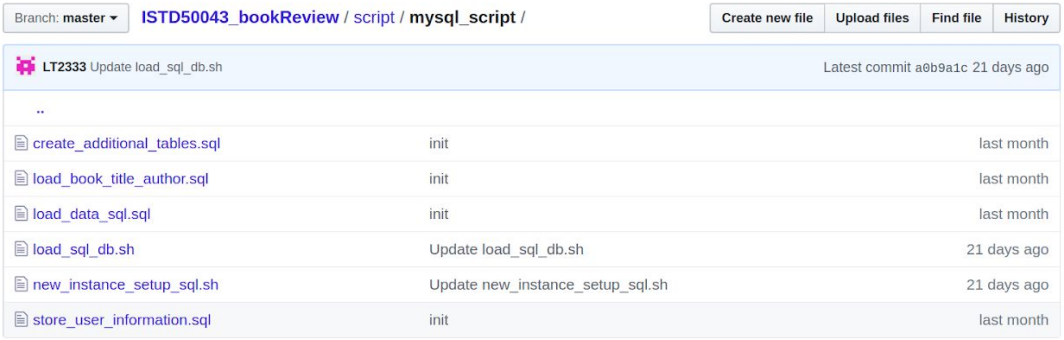
..		
analytics.sh	Update analytics.sh	3 hours ago
pearson_cal.py	Add files via upload	22 hours ago
setup_masternode.py	Add files via upload	17 hours ago
tfidf_cal.py	Add files via upload	22 hours ago



Branch: master ▾ ISTD50043_bookReview / script / mongo_script / Create new file | Upload files | Find file | History

LT2333 Update set_up_mongo.sh — Latest commit fbe9d0d 2 days ago

..		
assign_best_seller.py	init	last month
mongo_util.js	init	last month
set_up_mongo.sh	Update set_up_mongo.sh	2 days ago



Branch: master ▾ ISTD50043_bookReview / script / mysql_script / Create new file | Upload files | Find file | History

LT2333 Update load_sql_db.sh — Latest commit a0b9a1c 21 days ago

..		
create_additional_tables.sql	init	last month
load_book_title_author.sql	init	last month
load_data_sql.sql	init	last month
load_sql_db.sh	Update load_sql_db.sh	21 days ago
new_instance_setup_sql.sh	Update new_instance_setup_sql.sh	21 days ago
store_user_information.sql	init	last month

Application Features

#all web UI screenshots can be viewed in appendix

1. **Home Page**
   Users can view the highest ranked books on the homepage for the most popular categories and access other pages like booklist, their own data-logs of previous usage.
2. **Login**
   Logged in as a normal user, user could see his own book viewing history; Logged in as an admin user, user could see most viewed books of all users and log record including 1. web traffic summary of the month in line plot 2. Web traffic distribution in different time in different day of the week in the form of heat map (available in last week history, all history and a demo heat map of dummy log data).
3. **User and Admin Accounts**
   There are two types of accounts. User accounts allow the user to leave reviews on a book, while only Admin accounts can access the Add Book page in addition to the features available to a User account. Without a User account, one can only browse book information and search for books.
4. **Book Information and Review Page**

Book information like author, title, categories could be available. User after login could make comments and give a rating to this book. Ratings from all users will be collated and shown as the overall rating of this book.

5. **Add Book**
   Admin account can access the Add Book page from the homepage. On this page, Admin accounts add more books to the database with manual input of book attributes such as Title, ASIN number, pook price and more.
6. **Search**
   All users can search for books based on title, category, author, or ASIN number. The search functions are available on the homepage, and also in the top navigation bar in most other web pages.
7. **Book list catalogue**
   This page shows the full catalogue of books distributed in pages and sorted by category arranged in alphabetical order. One can access books of a certain category by choosing one after hovering over the alphabet buttons under the Category heading, or by clicking on the bolded category tags under each image of a book.
8. **Tags**
   The category tags in the Book list menu are clickable to automatically search for books of a certain category.
9. **Lazy loading**
   Efficiency and speed of our app was improved using lazy loading design pattern (deferring initialization of an object until the point at which it is needed) Therefore, our “booklist” page does not fetch all 400,000 books at the same time. It only loads 1000 books at a time, making our page return results much faster.

Project Architecture

#we already have some users and their faked activity records
#all admin details that are currently present

Frontend

Web Application

We used Flask, a lightweight WSGI (Web Server Gateway Interface) web application framework to build our app. It is designed with the quick and easy ability to scale complex applications.

The files for our application are present in the `src` folder on github. `mongoService.py, SQLservice_User.py, SQLservice.py` are the main files that connect with the backend. These files contain functions to fetch our db instance and collection (table). In `mongoService.py`, we create a connection to the database present on the ec2 instance using `MongoClient`. In `SQLservice_User.py` and `SQLservice.py`, we use `mysql.connector` to connect with our database and wrote functions to fetch the data in the format we need.

These functions are further used in `app.py` to send data from the database over to the front end. `app.py` contains the main code to render all the HTML templates present in the static folder.

Scrapper

Due to limitation of provided data of book metadata, most of the authors and titles are not available. Hence we have scraped information from amazon directly. How we conducted the scrapping is at scrapper.py and sample scraping result is at scrap_bookinfo_sample.csv under src/scraper directory

Back End

Production System

- ServerServer

We hosted our app on an ec2 instance: Before git cloning the web github repository, we output all the requiring libraries and corresponding version in requirements.txt. Then it will install the library accordingly and then run the flask app in the ec2 instance.
Due to the dynamic ip of mySQL and MongoDB server that we just created from automation script, we cannot fix them in the app's code. Therefore, we encoded them into environment variables 'LC_MONGO_IP' and 'LC_MYSQL_IP''. After creating the instances, we will pass the ip address when execute the ec2 commands as a temporary environment dictionary.

- Mongodb

Our MongoDB server is hosted on another separate EC2 instance, which allows our production server to write and read documents. Within the MongoDB server, there are two MongoDB databases, one named book-metadata, the other is book-log. The book-metadata stores the json file with information for all books including their title, author, related books, price and so on. We did some simple preprocessing and refinement for our data, including getting book titles through web crawling for books without titles and so on. All metadata about the books are stored in a collection of the database named metadata. Book-log database contains a collection called log which stores the log information generated from our production server, which records the query timestamp, username, query type, etc.

- SQL

We have created 2 mysql databases, one is for all the review data, another one is for user management.

    - *Data Processing*

We loaded the data according to the requirements and the datatype as shown below, and created 2 additional tables for faster access, '*mostRated'* and '*highestAvgScore*''.
'*mostRated'* returns the top20 books that rated by most number of users;
'*highestAvgScore*'' returns the top20 books that have the highest ratings.

```
mysql> desc reviews;
+---------------+--------------+------+-----+---------+-------+
| Field         | Type         | Null | Key | Default | Extra |
+---------------+--------------+------+-----+---------+-------+
| idx           | int(11)      | YES  |     | NULL    |       |
| asin          | char(10)     | NO   |     | NULL    |       |
| helpful       | text         | YES  |     | NULL    |       |
| overall       | int(11)      | YES  |     | NULL    |       |
| reviewText    | varchar(8000)| YES  |     | NULL    |       |
| reviewTime    | text         | YES  |     | NULL    |       |
| reviewerID    | text         | YES  |     | NULL    |       |
| reviewerName  | text         | YES  |     | NULL    |       |
| summary       | text         | YES  |     | NULL    |       |
| unixReviewTime| text         | YES  |     | NULL    |       |
+---------------+--------------+------+-----+---------+-------+
```

- *User Management*
    - Due to security reasons, we **encrypted** all the users' passwords by using MD5 as shown below.
    - To distinguish different users, we use 'isadmin' column to indicate its identity. If isadmin is 1, the user is an **administrator**, otherwise, he is a normal user.

```
+----+----------+----------------------------------+---------+
| id | username | password                         | isadmin |
+----+----------+----------------------------------+---------+
|  1 | Ainul    | e10adc3949ba59abbe56e057f20f883e |       1 |
|  2 | Jiankun  | e10adc3949ba59abbe56e057f20f883e |       1 |
|  3 | Pengfei  | e10adc3949ba59abbe56e057f20f883e |       1 |
|  4 | Yunyi    | e10adc3949ba59abbe56e057f20f883e |       1 |
|  5 | Shanshan | e10adc3949ba59abbe56e057f20f883e |       1 |
|  6 | Nashita  | e10adc3949ba59abbe56e057f20f883e |       1 |
|  7 | Lutong   | e10adc3949ba59abbe56e057f20f883e |       1 |
|  8 | test1    | e10adc3949ba59abbe56e057f20f883e |       0 |
+----+----------+----------------------------------+---------+
```

Analytics System

- General architecture of our HDFS
  We installed Hadoop v2.7 for our distributed file system and spark v2.4.4. Our HDFS architecture is one of the following, based on the user's input when generating the clusters:
    1. 1 master and 1 slave (2 nodes)
    2. 1 master and 3 salve (4 nodes)
    3. 1 master and 7 slave (8 nodes)

- Calculating Pearson correlation between price and average review length

    - All of the data access, data processing and then calculation of Pearson correlation is done within an instance of the PearsonCorrelationCalculator object class.

    - When an instance of the PearsonCorrelationCalculator object class is created, a PySpark session is initialised, along with attributes to store the

processed data (average review length and book price of the corresponding ASIN) and value relating to the Pearson correlation.

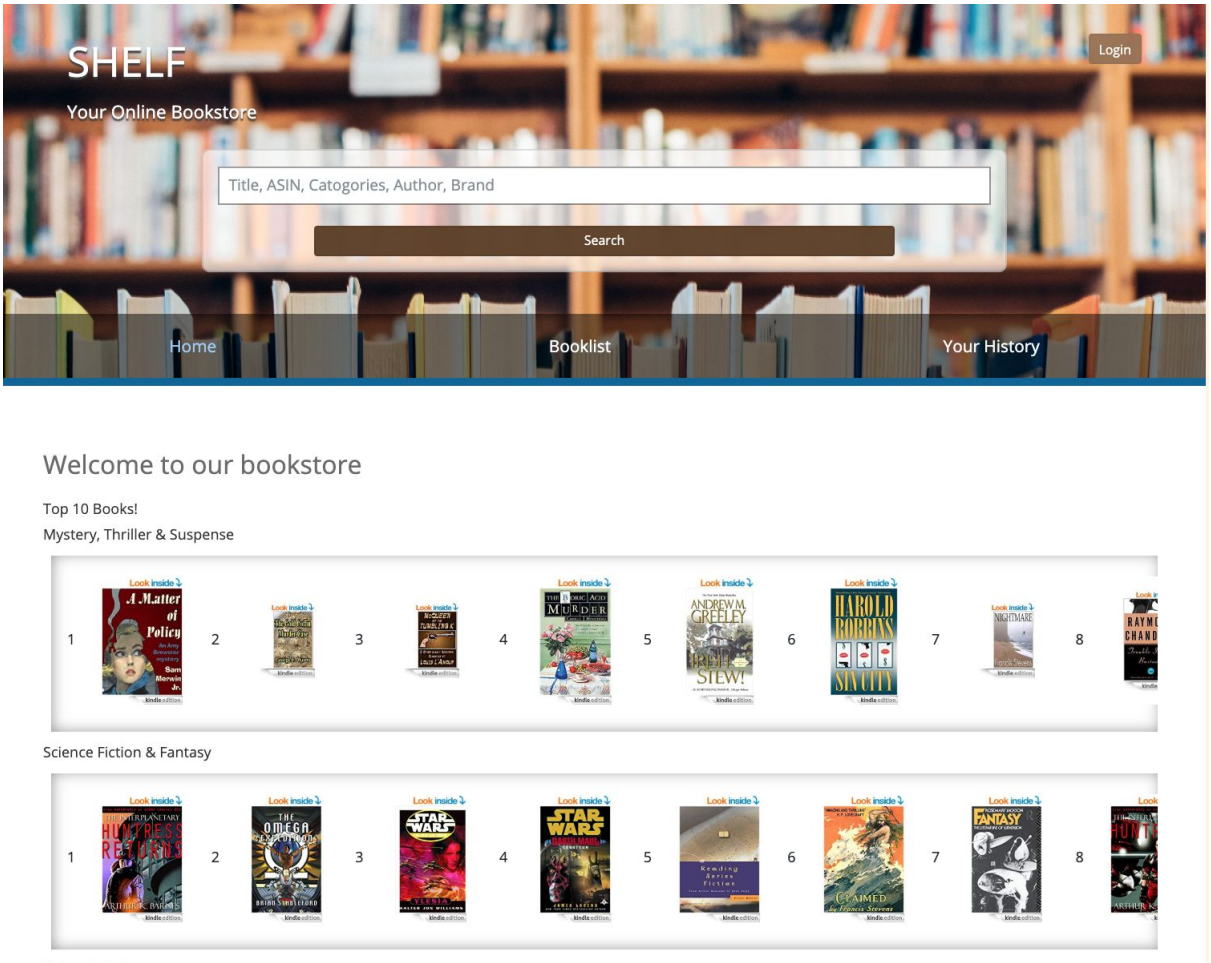
- The get_price_and_average_review_length method takes in the paths of the files containing book metadata and book reviews from Amazon Kindle (or local copies made on 14 December) by default. The ASIN and corresponding book prices are extracted from the book metadata, and the average review length of a book is also calculated for each ASIN with at least one review. These values are saved in an RDD with each Row containing ASIN number, book price and average review length, and the RDD saved to the Calculator's price_ave_review_len_rdd attribute.
- The calculate_pearson_correlation method calls for the price_ave_review_len_rdd and calculates the Pearson correlation between book price and average review length in a map-reduce fashion:
    - Using formula for Pearson correlation

$$r = r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}}.$$

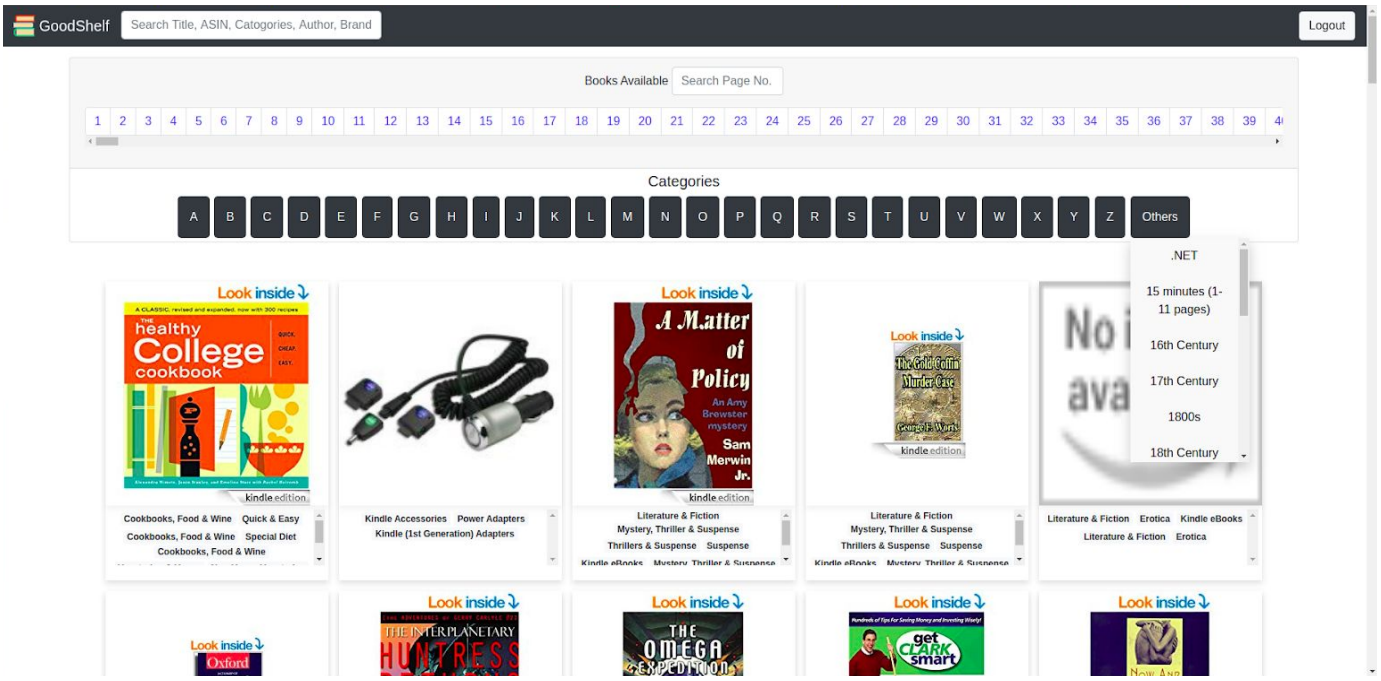
      we created the following map-reduce tasks:
        - (purple) map average_review_length x price
        - (red) map average_review_length [extract from RDD]
        - (orange/yellow) map square of average_review_length
        - (blue) map square of book price
        - (green) map book price [extract from RDD]
        - Each corresponding reduce task calculates the sum of each map separately i.e. (purple) sum of all *average_review_length x price*
    - The final step of finding the Pearson correlation is combining the outputs of the above map-reduce tasks into the formula. The calculated correlation value is saved to the Calculator's pearson_correlation attribute for future calling, and printing it to the console.
- **The default calculated Pearson correlation value is 0.023.**

Appendix

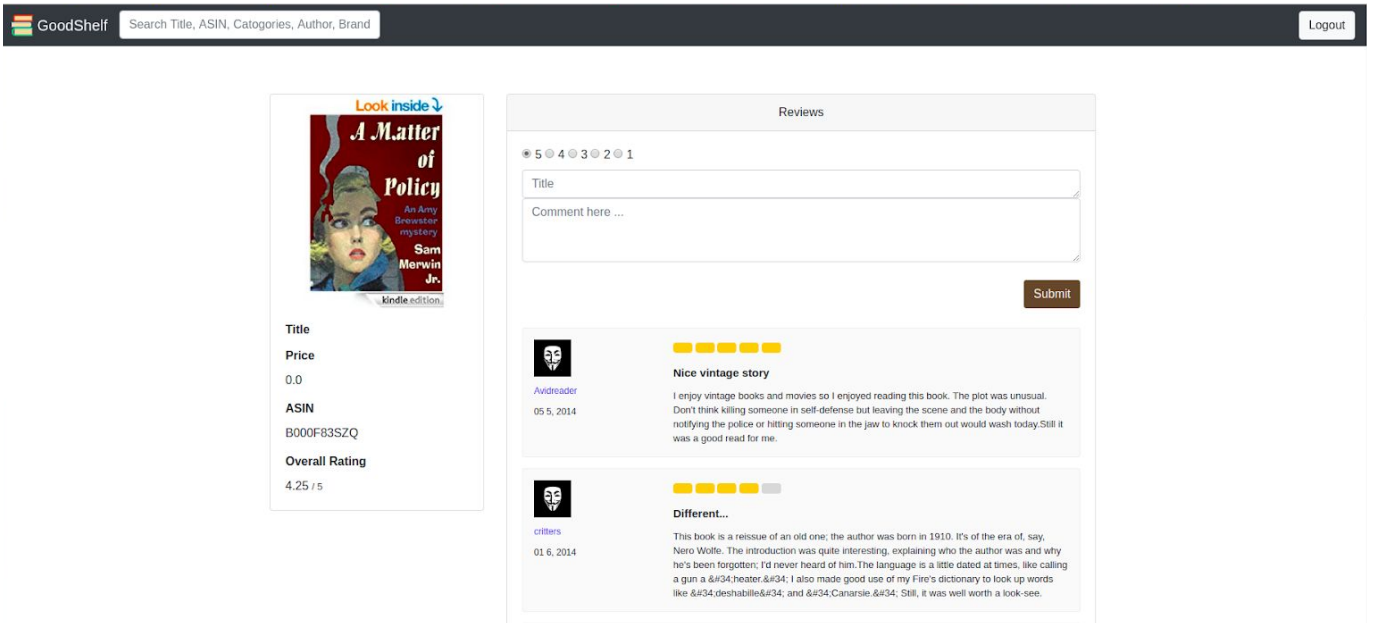
Home Page page screenshot:



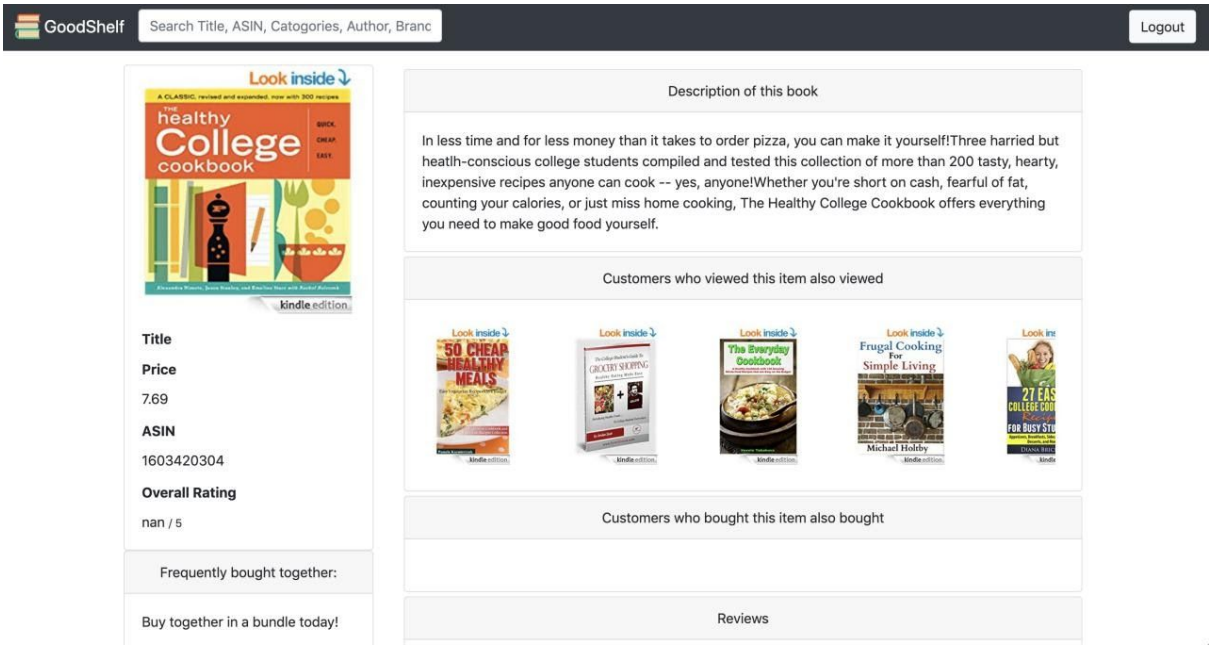
Booklist page screenshot:



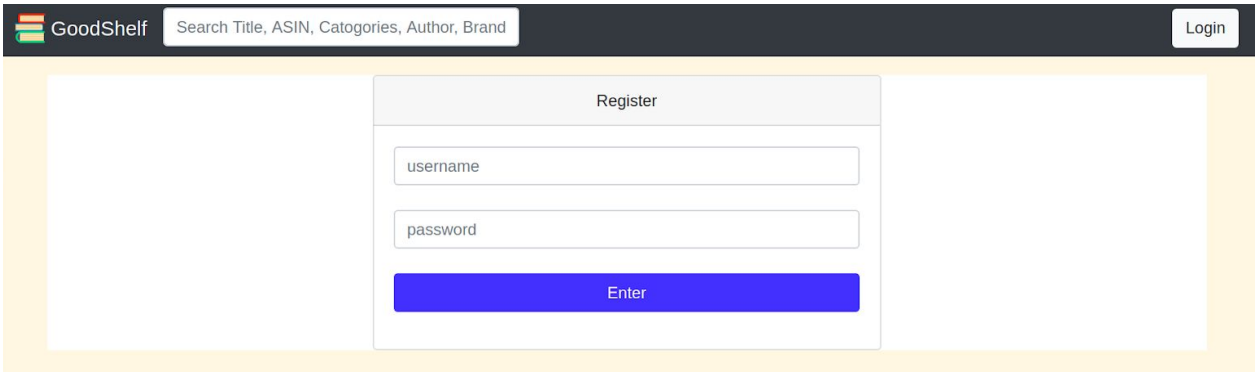
Each book info page screenshot:



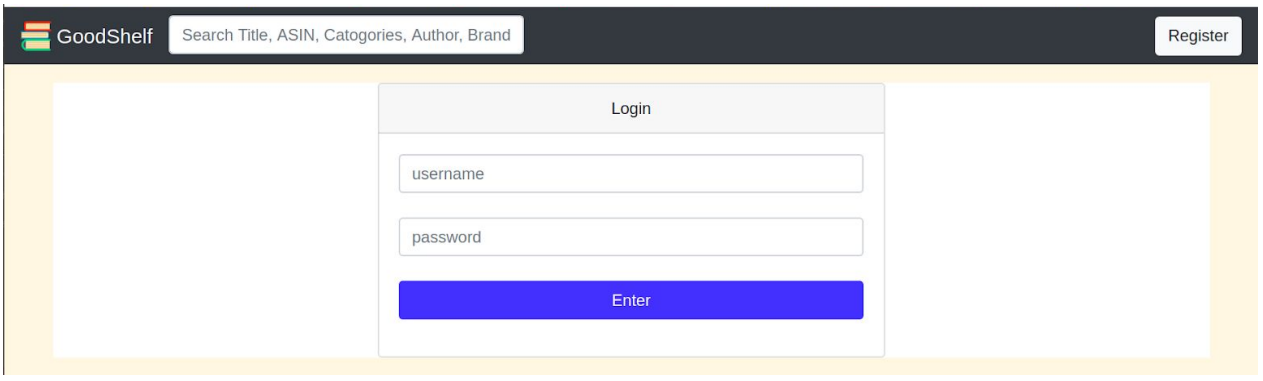
Based on the log record, some books contains “Customers who viewed this item also viewed” book record and “Customers who bought this item also bought” book record.



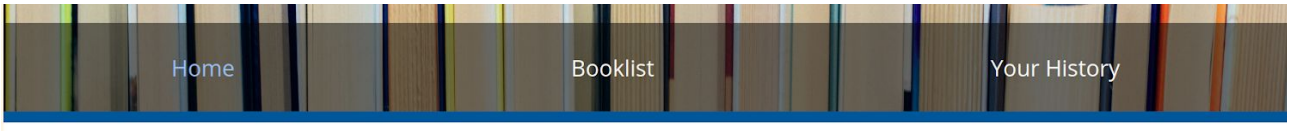
Login page screenshot:



Register page screenshot:

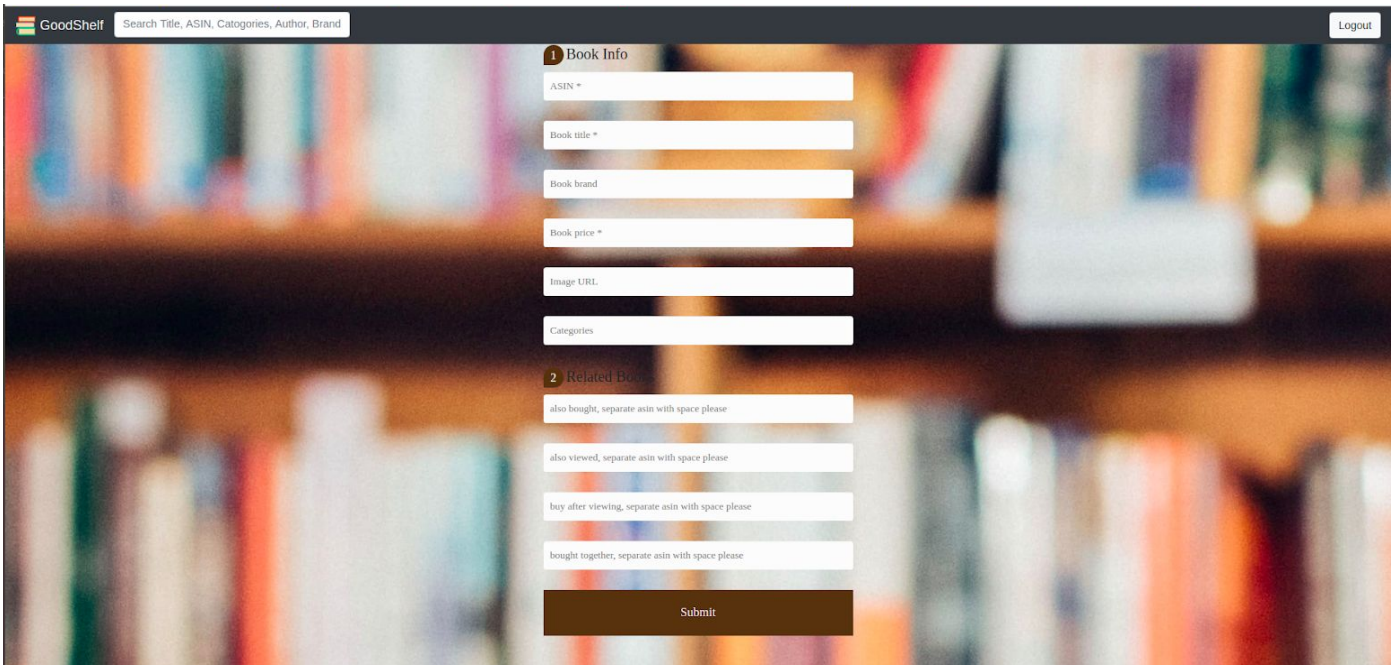


If you login as a normal user, the home page has the following access:



If you login as an admin user (e.g. username:Yunyi password:123456), the home page has the following access: (adding Add book function for Admin and can see all the log from

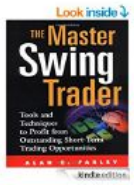
Add book page screenshot:



Log Record page screenshot:

Log Record

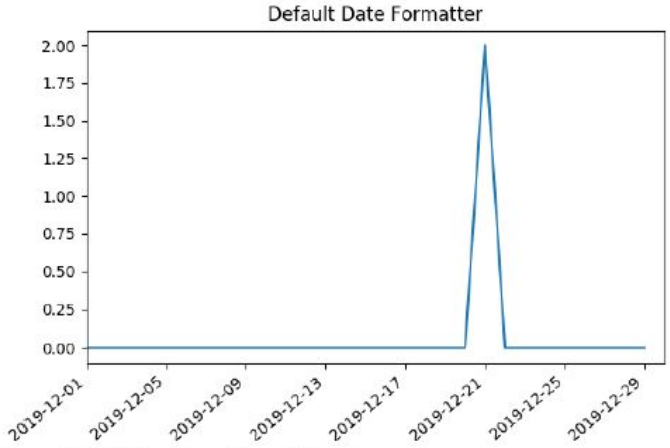
Most Viewed books

1 

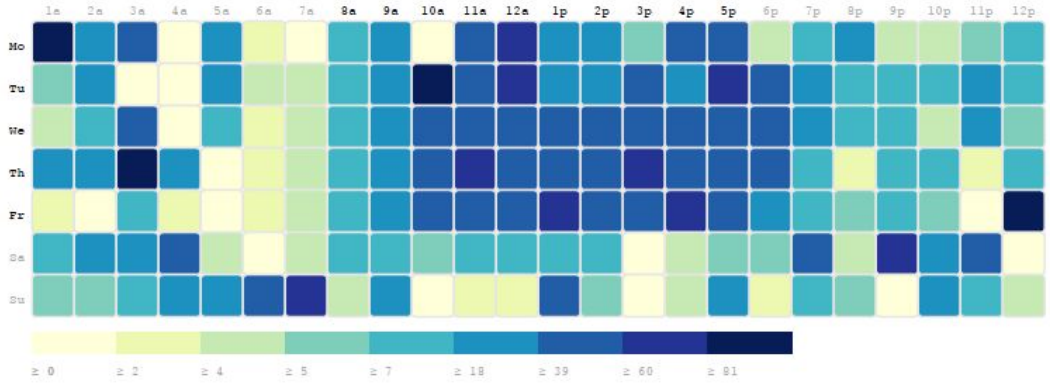
2 

Web Traffic

log frequency over the last month



Log Distribution in a Week



last_week_history | all_history | demo

Demo option is the heat map of fake log record(due to new instance construction, logs are not sufficient for a good graphical demonstration).